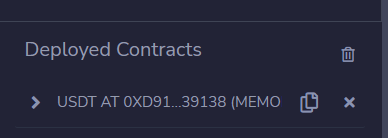
**周亚男 2020131062 202班**

**实验内容**

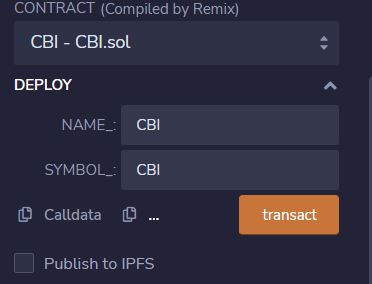
1. 准备好两个账户   
账户A： 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4  
账户B： 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

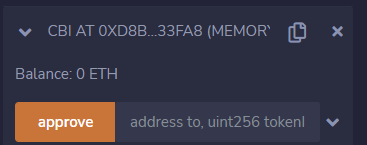
2. 使用账户A部署 USDT 合约；

0xd9145CCE52D386f254917e481eB44e9943F39138



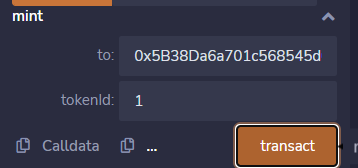
3. 使用账户A部署 CBI 合约

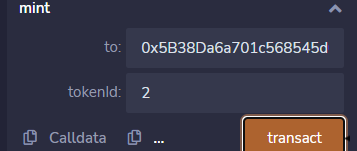




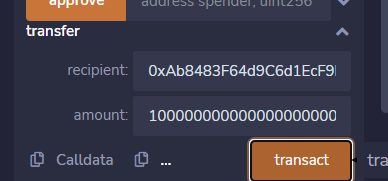
合约地址为：0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8

4. 使用账户A mint 2个nft

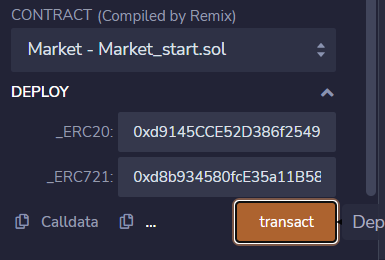


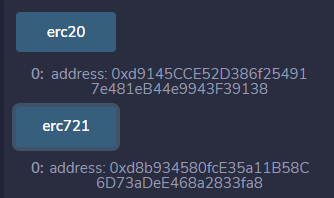


5. 账户A转给账户B 1000个USDT；



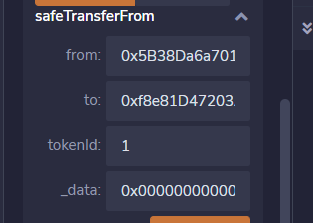
6. 部署Market合约



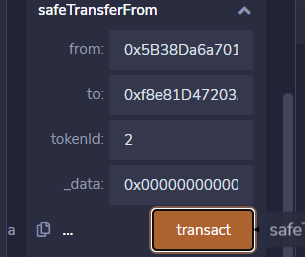


合约地址为：0xDA0bab807633f07f013f94DD0E6A4F96F8742B53

7. 账户A把tokenid为1的nft放到市场，定价100USDT



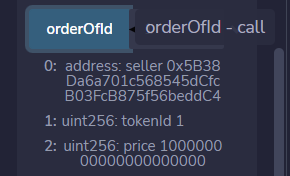
8. 账户A把tokenid为2的nft放到市场，定价200USDT



（想一想，上架到市场为什么用safeTransferFrom？data携带的 是什么信息？）

Safetransferfrom是安全传输，在调用此函数时，会触发前面的onerc721recieved（）来验证这笔交易是否合法正确。Data就是用bytes类型表示的价格。

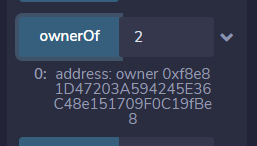
9. 在Market合约上查询tokenid为1的订单信息；



10. 在Market合约上查询tokenid为2的订单信息



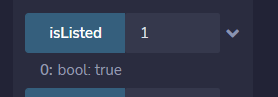
11. 在CBI合约查询tokenid为2的ownerOf



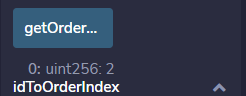
这个owner是谁？

Market这个合约账户

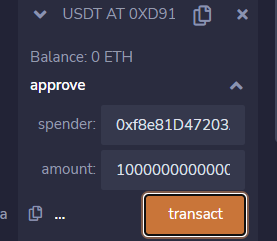
12. 在Market合约调用isListed(1)，再次确认tokenid为1 的nft上架情况



13. 调用Market合约的getOrderLength，查询上架nft订单总量

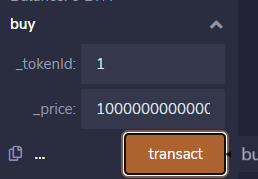


14. 使用账户B，在USDT合约对Market合约进行approve

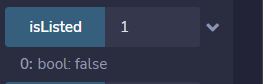


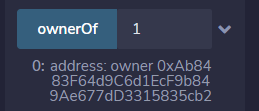
想一想，这一步是必须的吗？如果不进行approve呢？

15. 使用账户B，支付100USDT，购买tokenid为1的nft

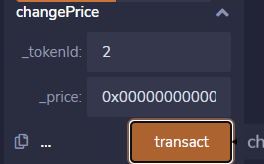


16. 在CBI合约查询ownerof（1），是不是账户B？





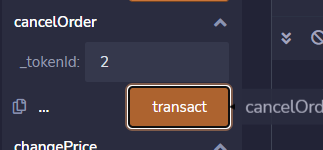
17. 使用账户A修改tokenid为2的nft价格为300 USDT；



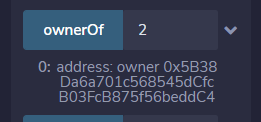
18. 在Market合约查询tokenid为2的订单信息



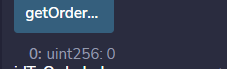
19. 使用账户A，在Market合约下架tokenid为2的nft；



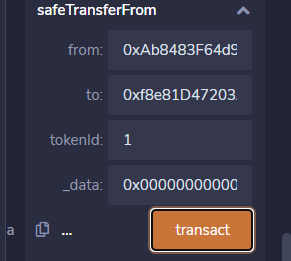
20. 在CBI合约查询tokenid为2的ownerOf；是不是账户A？



21. 在Market合约查询订单总量getOrderLength

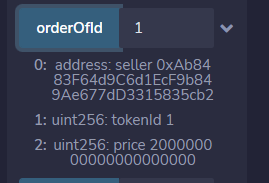


22. 使用账户B，在Market合约上架tokenid为1的NFT，价格为 150USDT；



23. 在Market合约查询tokenid为1的nft订单信息。

因为测试老师给的第三个数据有错误，这里采用的第二个数据，结果不影响。



**思考题：**

1. 为什么要把placeOrder方法做成internal（加分项）

在上架前必须要检验转nft是否合法，否则直接用这个函数，就可以随便转且不考虑任何后果。用safetransferfroom时会调用onerc721recieved去检验，此函数再去erc721合约中调用其他的函数进行转nft。

1. 解释toUint256的实现原理；（加分项

汇编。

**源码：**

// SPDX-License-Identifier: MIT

pragma solidity 0.8.4;

import "./interfaces/IERC721Receiver.sol";

import "./tokens/ERC721.sol";

import "./tokens/ERC20.sol";

import "./utils/SafeMath.sol";

// 0x0000000000000000000000000000000000000000000000056BC75E2D63100000

// 0x00000000000000000000000000000000000000000000000AD78EBC5AC6200000

// 0x000000000000000000000000000000000000000000000000821AB0D4414980000

contract Market is IERC721Receiver {

    ERC20 public erc20;

    ERC721 public erc721;

    bytes4 internal constant MAGIC\_ON\_ERC721\_RECEIVED = 0x150b7a02;

    struct Order {

        address seller;

        uint256 tokenId;

        uint256 price;

    }

    mapping(uint256 => Order) public orderOfId; // token id to order

    Order[] public orders;

    mapping(uint256 => uint256) public idToOrderIndex;

    event Deal(address buyer, address seller, uint256 tokenId, uint256 price);

    event NewOrder(address seller, uint256 tokenId, uint256 price);

    event CancelOrder(address seller, uint256 tokenId);

    event ChangePrice(

        address seller,

        uint256 tokenId,

        uint256 previousPrice,

        uint256 price

    );

    constructor(ERC20 \_erc20, ERC721 \_erc721) {

        require(

            address(\_erc20) != address(0),

            "Market: ERC20 contract address must be non-null"

        );

        require(

            address(\_erc721) != address(0),

            "Market: ERC721 contract address must be non-null"

        );

        erc20 = \_erc20;

        erc721 = \_erc721;

    }

    function buy(uint256 \_tokenId, uint256 \_price) external {

        address buyer = msg.sender;

        Order memory targetOrder = orderOfId[\_tokenId];

        uint price = targetOrder.price;

        address seller = targetOrder.seller;

        require(\_price >= price, "price not enough");

        // ERC20打钱

        erc20.transferFrom(buyer, seller, \_price);

        // ERC721转移给买家

        erc721.safeTransferFrom(address(this), buyer, \_tokenId);

        // 下架 NFT

        removeListing(\_tokenId);

        emit Deal(buyer, seller, \_tokenId, price);

    }

    function cancelOrder(uint256 \_tokenId) external {

        Order memory targetOrder = orderOfId[\_tokenId];

        address seller = targetOrder.seller;

        require(msg.sender == seller, "Only seller can cancel order");

        removeListing(\_tokenId);

        erc721.safeTransferFrom(address(this), msg.sender, \_tokenId);

        emit CancelOrder(seller, \_tokenId);

    }

    function changePrice(uint256 \_tokenId, uint256 \_price) external {

        Order storage targetOrder = orderOfId[\_tokenId];

        address seller = targetOrder.seller;

        require(seller == msg.sender, "only seller can change price");

        uint previousPrice = targetOrder.price;

        targetOrder.price = \_price;

        emit ChangePrice(seller, \_tokenId, previousPrice, \_price);

    }

    function onERC721Received(

        address \_operator,

        address \_seller,

        uint256 \_tokenId,

        bytes calldata \_data

    ) public override returns (bytes4) {

        uint \_price = toUint256(\_data, 0);

        placeOrder(\_seller, \_tokenId, \_price);

        return MAGIC\_ON\_ERC721\_RECEIVED;

    }

    function isListed(uint256 \_tokenId) public view returns (bool) {

        return orderOfId[\_tokenId].seller != address(0);

    }

    function getOrderLength() public view returns (uint256) {

        return orders.length;

    }

    function placeOrder(

        address \_seller,

        uint256 \_tokenId,

        uint256 \_price

    ) internal {

        Order memory newOrder = Order( \_seller, \_tokenId, \_price );

        idToOrderIndex[\_tokenId] = getOrderLength();

        orders.push(newOrder);

        orderOfId[\_tokenId] = newOrder;

        emit NewOrder(\_seller, \_tokenId, \_price);

    }

    function removeListing(uint256 \_tokenId) internal {

        uint index = idToOrderIndex[\_tokenId];

        Order memory lastOrder = orders[getOrderLength() - 1];

        orders[index] = lastOrder;

        idToOrderIndex[lastOrder.tokenId] = index;

        orders.pop();

        orderOfId[\_tokenId].seller = address(0);

    }

    // https://stackoverflow.com/questions/63252057/how-to-use-bytestouint-function-in-solidity-the-one-with-assembly

    function toUint256(bytes memory \_bytes, uint256 \_start)

        internal

        pure

        returns (uint256)

    {

        require(\_start + 32 >= \_start, "Market: toUint256\_overflow");

        require(\_bytes.length >= \_start + 32, "Market: toUint256\_outOfBounds");

        uint256 tempUint;

        assembly {

            tempUint := mload(add(add(\_bytes, 0x20), \_start))

        }

        return tempUint;

    }

}